KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculty of Business and Economics

The use of the concept of event in enterprise ontologies and requirements engineering literature

Pieter Hens, Monique Snoeck, Geert Poels and Manu De Backer

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

KBI 0909

The use of the concept of event in Enterprise Ontologies and Requirements Engineering literature

Pieter Hens[1], Monique Snoeck[1], Geert Poels[2], Manu De Backer[3]

[1] K.U.Leuven, Dept. of Decision Sciences and Information Management,
Naamsestraat 69, B-3000 Leuven, Belgium
{Pieter.Hens, Monique.Snoeck}@econ.kuleuven.be

[2] Universiteit Gent, Dept. of Management Information and Operations Management
Tweekerkenstraat 2, B-9000 Gent, Belgium
Geert.Poels@ugent.be

[3] Universiteit Antwerpen, Dept. of Management Information Systems
Prinsstraat 13, B-2000 Antwerpen, Belgium
Manu.DeBacker@ua.ac.be

Abstract

The concept of event is used in a lot of meanings. It can be the possible outcome of doing something (probability theory), it can be a business transaction (accounting), or just a plain happening. In software engineering, the concept of event is also used a lot. It is used to accomplish loose coupling between software components or to realise interaction between different services. There is however not a consensus on the meaning of 'an event'. In enterprise ontologies, an event is defined as a happening at one point in time, or as an activity which takes time to complete. In requirement engineering, the same different uses can be found, together with an event as a request for something that needs to be done. These differences can also be found in implementation. All these distinct purposes of the word event make it difficult to integrate and use different requirement engineering techniques. Comparison or transformations between models drawn in different grammars is impossible because of the ambiguity of the concept of event. We define three meanings for an event that are used by enterprise ontologies and requirement engineering techniques: an achievement (happening at one point in time), an activity (happening over time) and a request (a demand for something that needs to be done). We also identify a missing link between real economic events, the events defined in the requirements model and the events used in implementation.

1 Introduction

The concept event can have a lot of meanings. When used in different contexts, different definitions are given to 'an event'. In probability theory an event is one or more of the possible outcomes of doing something [23]. In accounting an event indicates a business transaction that requires a journal entry [31]. In the minds of most people an event is something more general and vague. It is something that happens, an occurrence.

Not only does the word event have different connotations over different contexts. In the same setting, the vague term event is also used for different purposes. This is the case in information system design. An event can be an observable happening in a computer system, but it can also represent a request for something that needs to be done. Both in requirements engineering and in the implementation of the information system the concept of event is used for different purposes. This can lead to a lot of confusion. An event used in one technique isn't necessarily the same as an event used in another technique. Alongside the problem of different uses of events, is the problem of vague, non-complete definitions of the concept of event in different requirements engineering and software development techniques. Most of the time it isn't clear what is meant with 'an event', it is up to the reader or user to fill in the gaps.

Vague definitions and different uses of the concept of event make it all the more difficult to build an (event based) information system from start to finish.

2 Problem description

Lack of well defined, clear definitions When reading about requirements engineering, the word 'event' is often used. Especially in requirement engineering techniques that want to build dynamic, fast adaptable systems. The use of an event in one technique isn't the same as an event in another (and can be completely different). Well defined definitions of the concept event are missing. This makes it hard for a modeller using a technique to know what is meant with 'an event', it is confusing and ambiguous. There also isn't a well defined super-concept event that can be used in all information system modelling techniques.

Mapping between different models, implementation and real world business events The different uses of the word event make it difficult to compare modelling techniques and perform transformations between models. For example, transformation of a platform independent model to a platform specific one, like it is done in Model Driven Architecture [19]. Events in the modelling technique can't be mapped to events in the implementation of the information system.

Another mapping that becomes difficult is the mapping between events defined in the requirements modelling technique and real business events. With business events we mean real-world happenings of interest to the enterprise, business activities.

In this document we give a classification for the different event interpretations that exists in the requirements engineering literature. An event in a modelling technique can be classified in one (or more) of these interpretations. This makes it a lot easier to see what the authors of these techniques meant with an event and makes the mapping between different techniques less complex.

3 Research method

Below a classification is given for the different uses of an event in requirements engineering and enterprise ontology literature. We used a detailed literature review to come to this classification. Requirements techniques and enterprise ontologies that use the notion 'event' were used to conduct this literature review.

Enterprise ontologies were also included in the study because they give a solid theoretical base on which requirements models can be build. If the concept of event is ill defined in the ontology,

it will also show in the requirements model. Ontologies can also be used to check the validity of a requirements model, this also means that the definitions given in an ontology should be correct and unambiguous (for it to serve as a validation technique).

Research questions

RQ1. How is the concept of event used in different requirements engineering techniques?

RQ2. Is there a consistent use of the concept of event between the different requirement engineering techniques?

RQ3. How well defined is the concept of event in the requirement techniques specifications, in enterprise ontologies and in the foundational ontologies?

Source selection

We analysed the use of the concept event in the specifications of different requirement engineering techniques, as well as in enterprise ontologies (qualitative data). Data collection was done by:

- Searching with google scholar, the ACM digital library and the IEEE computer society;
- Browsing with cetus links; and
- Crawling the references of the papers and books found with the above techniques.

Keywords used to search for papers were: 'requirements modelling', 'enterprise ontology', 'process modelling', 'dynamic' and 'event'.

Study selection

Requirements engineering techniques were only added to the data set, when the technique is used for requirements modelling (state based or activity based) and when the technique defines and uses the concept 'event' in its model.

Enterprise ontologies and foundational ontologies were added to the data set if they defined the concept event in the ontology.

Analysis

Next, the gathered data was *analysed*, *notes* were taken and similar uses of the concept event were grouped together. This grouping led to the defined *categories*. Afterwards, properties were derived for each category, using the data classified into each category.

4 Differences

4.1 Event interpretations

When looking at the use of the concept of event in requirements engineering techniques and enterprise ontologies, three different categories can be defined. An event is used in the context of an activity, a happening at one point in time or a request.The latter two concepts are used to model interaction between different entities: interaction between activities (e.g. in the process flow) or interaction between different agents (e.g. buyer and seller). Because we only look at the uses of 'events' in enterprise ontologies and requirements engineering techniques, the defined concepts are positioned in the first three layers of the Zachman framework (see figure 1). Each of these event definitions is further explained below.

	What (Data)	How (Function)	Where (Locations)	Who (People)	When (Time)	Why (Motivation)
Scope (Contextual)	Activity (As data concept)	Activity (As an executable action)		Request	Achievement	
Enterprise Model (Conceptual)						
System Model (Logical)						
Technology Model (Physical)						

Figure 1: Events in the Zachman framework

4.1.1 Activity

An event can be used in the meaning of an activity. The event is seen as something that takes place. It is an event in the common sense of the word, a happening. It is an action, the execution of something. This implies that an activity has a time duration. Time is needed to process the activity.

An activity can also be observed, it can be detected that the action is taking place. The action can be a real world action or an action in the information system. An activity also has an outcome, it changes the state of the real world and by consequence can change the state of the information system.

Some examples of activities are:

- a person with a gun walks through a metal detector
- a user of a security application presses the right mouse key
- an alarm goes off
- a car salesman creates a new order
- a borrowed book is returned
- a travelling salesperson is driving from the south to the north pole
- an update of a record in a database

All of these examples are observable entities, which all take time to complete. The click on a mouse button lasts a couple of milliseconds, while the creation of a new order can last a couple of minutes and the travelling salesperson can be en route for several years. The outcome of the activity 'a borrowed book is returned' is that the book is back in the library and available for someone else. The outcome of a 'car salesman creates an order' is that the order is created so that the client will receives his car. We assume that an activity will, somehow, always change the state of the real world (the outcome). For example, when a salesman creates a new order, the state of the world changes from `X` to `X+new-order-created`. Information systems are built to mimic or record changes in the real world. If we build an information system that mirrors the portion of the real world where the change happened (e.g. car supplier), a change of state will also occur in the information system.

The above examples bring up the concept of *granularity of events*. The return of a borrowed book has a more coarse grained granularity than the click on a mouse button. Fine grained means a high level of detail, and that a lot of information is available about the event. An activity can encapsulate other activities of finer granularity. The creation of a new order by a car salesman requires the input of data in a software application and (automated) communication with car suppliers. Data input by a human user requires key- and mouse-presses. A key press requires the movement of the users finger, ... This can also be seen as composition of events. Each event can be composed of other finer grained events.

Event as an activity is used in Merode (see section 5.2.4), REA (see section 5.1.5) and in several foundational ontologies (see section 5.1).

Properties

An event as an *activity* has the following properties:

- It is an action, an execution, *the happening itself*
- An activity has *a duration time*
- An activity changes the state of the real world and it can change the state of the information system
- An activity can be observed
- Not used to model the interaction between entities

Activity as an executable action vs Activity as a data concept

When translating the requirements model to implementation, we need to make a distinction between the *real executable activities* and activities as a *data concept*. When using an activity in process modelling, the activity will be translated to real actions. This means that the activity describes actions, code that can be executed to accomplish something. The activity in the model can also be translated to a data concept, to the bill-of-material of the enterprise. This means that the concept will only describe the (real world) activity. It will describe the properties, relations of the activity. It won't implement dynamic procedures necessary to execute the activity.

Note that both concepts represent activities in the real business world (executed by humans or automated in the information system). This is the activity-event as described above. Only when a translation is made from the requirements model to the implementation of the information system, another distinction can be made: between activity as an executable action in the information system, or activity as a data concept in the information system.

An example of this distinction is the activity-event 'payment'. A payment can be implemented in the information system as all the actions that need to be done to accomplish this: lower the account balance of one party, increase the balance of the other party, ... When the payment event is implemented as a data concept, the payment will be modelled as an *object* describing it's properties: amount, payer, receiver, ... Remark that an activity-event can be implemented as a data concept *and* as a (set of) accompanying activities (like the payment-activity).

4.1.2 Achievement

Another use of the concept event is an event as something that happens *at one point in time*. We call this an achievement, corresponding the term defined in the DOLCE ontology [12]. An achievement-event can be the start of a process, the end of a process, the state change of an entity, the change in time (it's 12 o'clock), ... In general an achievement is *the* moment in time where a new state in the information system or the real world is reached. The change is instantaneous and is induced by a running activity, a sensor monitoring, ...

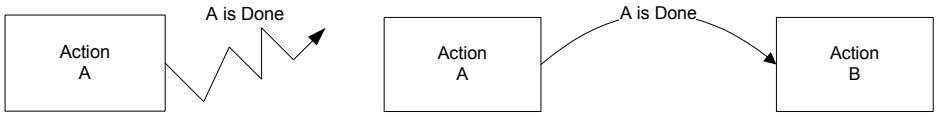
Notice the difference with an activity-event which has *a time duration*. The activity itself will need time to execute it's actions, but the start or end of the activity will be instantaneous. We can put an exact time on when this happened. For example: the walking of a person with a gun through a metal detector will take time, but the detection of the gun will happen at one point in time (e.g. September 12th, at 1:23:24pm), it is instantaneous. The achievement *is* the change in the information system, which is *induced* by an activity.

An achievement can be observed from within the information system and interested parties can react to this. The fact that this happening is instantaneous also implies that from the moment it is observed, the happening *happened* (past tense). The occurrence can't *be happening*, because it doesn't have a time duration. An achievement thus represents something that already happened, it can't be prevented or influenced by another process (from the moment it is observed).

An achievement event is the most common used event concept. When modelling an information system with events, most of the time achievement-events are used to indicate changes in the information system and reactions to these changes are modelled. The changes always indicate happenings at one point in time. For example, when we want to create a system that starts wiggling a baby's crib when the baby cries, we implicitly want to say that the crib should be wiggled when the baby *starts* crying. The start of the crying of the baby is a happening at one point in time, an achievement.

Of course a discussion can arise on when the baby is in a crying state (e.g. when she cries louder than 110dBA), but this change of state (not-crying to crying), whenever it happens, will happen instantaneously.

When modelling a process with achievement-events, it must be kept in mind that an activity that induces an achievement-event, will not know the process flow. The activity does something, and because of its doing another activity reacts to what has been done. However, the inducing activity has no knowledge of this. But, when looking at the process flow from the system point of view (or the system modeller point of view), the eventual reaction on the achievement-event is known. The modeller of the system knows the process flow, and which event(s) triggers which action(s) (see figure 2).



(a) How much the activity knows

(b) How much the system knows

Figure 2: Knowledge of achievement-events

From a technical perspective, achievement-events will be modelled with the help of notifications. A notification is a message that holds all the necessary info, describing the event. This message can be produced by the activity that induces the event and it can be consumed by one or several other processes or activities. This is described by Mühl [25].

Examples of achievement-events are:

- a person with a gun is detected
- right mouse key pressed
- new order created
- the order changes from state 'open' to 'confirmed'
- a database record is updated
- ...

Properties

An event as an *achievement* has the following properties:

- It is a *happening at one point in time*
- An achievement doesn't have a duration time, it is *instantaneous*
- As soon as the happening is observed, it *happened*, it therefor *can't be influenced or prevented*
- Used to model the *interaction* between entities
- The activity inducing the achievement has no knowledge of the process flow

4.1.3 Request

The third use of the concept event in behaviour modelling is the use of an event as a request to execute a certain action. It is a demand, a call for something that needs to be done. A request is specifically used to trigger another activity. This has as a consequence that the activity that sends out a request-event will have to know what has to happen next (it incorporates business logic). In the example of the metal detector: a request turn-on-the-alarm will be send out when a person with a gun walks through a metal detector. The 'howl the alarm' process will react to this request and start the alarm. The activity that triggered the request does know what happens next.

Unlike an achievement-event which indicates something that already happened, a request event indicates something that needs to be done. This means that rules and preconditions can be applied on the request. The requested action can still be refused when a precondition is not valid. For example,

a request for turn-on-the-alarm is only created when metal is detected and the person who walks through the metal detector doesn't have a security clearance (two preconditions).

Examples of request-events are:

· turn on the alarm
· check customer information
· create a new order
· send out the travelling sales person
· update a record in a databank
· ...

Note that these examples indicate the demand to execute a certain action, not the action itself.

Request-events are used in Syntropy (see section 5.2.2), Remora (see section 5.3.4), Merode (see section 5.2.4), Catalysis (see section 5.2.3) and Object Oriented SSADM (see section 5.2.5).

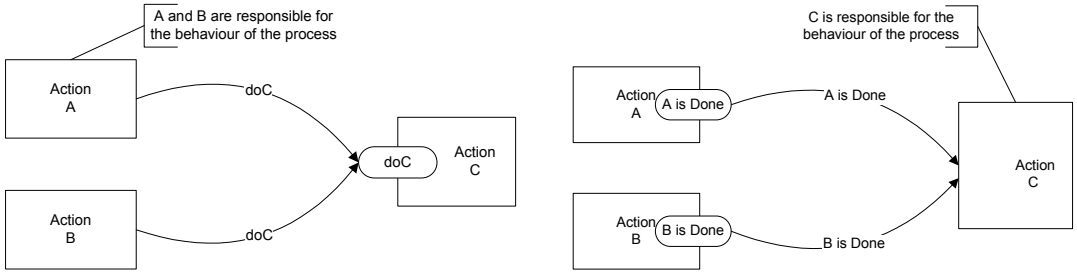
Properties

An event as a *request* has the following properties:

· It represents something that *needs to be done*
· *Preconditions* can be applied on requests
· A request doesn't have a duration time, it is *instantaneous*
· Used to model the *interaction* between entities
· Can't change the state of the information system

4.1.4 Achievement vs Request

Three different interpretations of 'an event' are described above. We argue that the concepts 'achievement' and 'request' are equivalent, but with their own characteristics. Though the two concepts are very different, a system modelled with requests can be transformed into a system modelled with achievement-events, and vice-versa. This can be done with the help of knowledge about the process flow.



(a) Process behaviour with a request

(b) Process behaviour with a notification

Figure 3: Request vs Notification

Suppose we want to model a process where an activity `C` should start after activity `A` executed or after activity `B` executed. This situation can be modelled with the help of request events or achievement events.

When this is modelled with a request event, `A` will send a request indicating that `C` must be executed. `B` will do the same. In this case `C` will only listen to one (request) event: `doC` and it will start it's execution when it receives this event (see figure 3a).

When the process behaviour is modelled with the help of achievement-events, `A` and `B` don't do anything special and just perform their actions. `C` will now listen to two events, one indicating that `A` is done and one indicating that `B` is done. `C` will start when it notices either of these two event (see figure 3b).

In both situations, the behaviour of the process flow is exactly the same. `C` will start after the execution of `A` or `B`. The difference between the two approaches is *a*) the location of the logic of the process flow (preconditions); and *b*) the level of control after an event happened.

Achievement

When achievement-events are used, the inducing activity doesn't know (and doesn't need to know) the next action in the process flow. It just performs it's actions and interested parties will notice this. The logic and the responsibility for handling the next step in the process flow lies with the listening activity. The listening activity will respond in the correct way to detected achievements. This means that when modelling with achievement events, no control over the process flow is needed for the inducing activity. It leaves all the responsibility and logic to the listening activity (if there is one). This activity will check the rules and preconditions to execute it's action. For example, the activity 'howl the alarm' will detect that a person with a gun walks through the metal detector, it will then check if the person has a security clearance, if not, it will turn on the alarm (all the logic is encapsulated in the listening activity).

Request

When using request events, the logic of the process flow is encapsulated in the requesting activity. This activity must know what has to happen next and compiles a request for the execution of the next action. This way the requesting activity keeps control over the flow, it is responsible for the next action. For example, a request for turn-on-the-alarm *is only sent out* when the person detected doesn't have a security clearance (the logic is encapsulated in the requesting activity). Staying in control also means that the requesting activity probably wants to be notified of the good or bad execution of the requested activity, and react accordingly.

4.1.5 Overview

In table 1 an overview is given for the different event interpretations and their properties.

		Activity	Achievement	Request
Interpretation				
	A happening with a duration time	X		
	A happening at one point in time		X	
	A demand			X
Execution Time				
	Instantaneous		X	X
	Time needed	X		
Outcome		X *		
Precondition possible		X		X
Representation in time (viewed from the events perspective)				
	Past		X	
	Future			X
	Present	X	X	
Used for interaction			X	X

* It is assumed that, somehow, an outcome is always present in the real world, while it is a possibility in the information system

Table 1: Overview of event interpretations

4.2 Other differences

Besides the use of the different event interpretations, there are also other differences found when events are used in requirements engineering.

Composition of events Some techniques allow that events can be specified into other, more detailed events, other techniques don't allow this. An example of this composition was given in section 4.1.1.

Level of abstraction of the use of events Some techniques require the modeller to think about very abstract events, while others define events with more detail. It is possible to compare the different modelling techniques with this scale. The scale is the abstraction level that the modeller has to think about, when defining events in the requirements engineering technique. Low on the scale means that the modeller is supposed to think about detailed, technical events, events that take care of the communication/interaction/orchestration of actions. High on the scale means that the modeller must think in a more abstract way, more about the real world business happenings and more independently from information system interactions. Another way to make this distinction is to say that high on the scale means that the events are recognisable by the business users, while low on the scale means that the events are gibberish for the business users.

Care must be taken when doing this. We compare the use of events *in general* in one technique with another. We don't compare the abstraction level of *individual* events. The latter is difficult and context dependent. For example, when comparing a technique that defines an event as something that is important to the business (has lots of business value), the event 'return of a book' *might* be more abstract than the event 'left mouse button click'. This is true in the context of a library, but it doesn't hold in the context of a website ad agency. The ad agency generates income per mouse click. A mouse click event in the context of the ad agency has a greater business value than the return of a book.

Dispatching of events When events are used to model interaction (achievements- and requests-events), it can be defined which entities will take part in the interaction, or who will catch certain events. Some techniques define this explicitly, others say that events are broadcast to every entity in the model. Even others don't use the concept of event to indicate the throw/catch mechanism, but see an event in conjunction with a simple one to one request/reply, like in BPEL (see section 5.4).

Events explicitly used for loose coupling Another difference is the answer to the question: 'Why are events used in the modelling technique?'. Events can be explicitly used to improve loose coupling and adaptability, which improves the alignment of business and ICT. In this approach events are used as a basic building block in the methodology. On the other hand, events can only be mentioned as a side note, a special technique that the modeller *could* use.

Events used to model interaction from outside the model In some techniques the concept of event is used to indicate activities (or achievements and requests) not performed by agents defined in the requirements model. Activities performed by agents in the model are simply referred to as actions, while activities from the environment, external agents, ... are referred to as 'events'.

5 Literature classification

In this section a description is given for requirements engineering techniques, enterprise ontologies and foundational ontologies that use events. Each technique is then classified in one of the four event interpretations.

The ontologies researched are DOLCE (section 5.1.1), UFO (section 5.1.2), BWW (section 5.1.3), the enterprise ontology (section 5.1.4) and REA (section 5.1.5).

In requirements engineering, events are used in two basic modelling areas. First, events are used in state machines, they help to model the behaviour of single entities, their states and state transitions. Events in state machines are used by UML (section 5.2.1), Syntropy (section 5.2.2), Catalysis (section 5.2.3) and Merode (section 5.2.4).

A second use of events is in process modelling or activity based modelling. Events are used to model and control the process flow. This is done in Event driven Process Chains (section 5.3.2), process-driven event based business object model (section 5.3.3), EDOC (section 5.3.5), remora (section 5.3.4), Dynamic Workflow Modelling (section 5.3.6), BPMN (section 5.3.7) and BPDM (section 5.3.8), Object Oriented SSADM (section 5.2.5), AOR (section 5.3.1) and in the implementation of the information system (section 5.4).

5.1 Ontologies

Ontologies in requirements engineering are used to provide a theoretical basis for information system models and grammars. They define concepts that should be integrated into a modelling language for it to be correct and complete. An ontology starts from the idea that any modelling grammar should be able to represent all things in the real world, of interest to users of the information system. An ontology can be used to evaluate an information system grammar. The ontology can indicate ontological incompleteness and ontological clarity of a grammar [38].

5.1.1 Descriptive Ontology for Linguistic and Cognitive Engineering

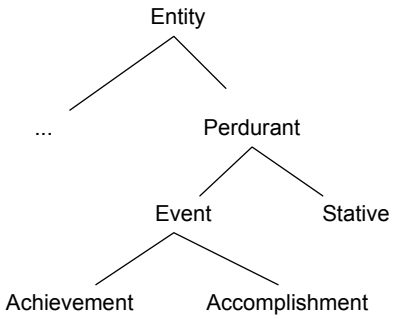


Figure 4: DOLCE event

DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [12] is a high level, foundational ontology (or upper level ontology). A foundation ontology is a domain wide ontology, it doesn't fit the needs of a specific community. It rather forms a foundation for other, more domain specific ontologies.

DOLCE also defines the concept event. According to DOLCE, an event is a perdurant[1] of the eventive type. Eventive means that a change, an eventuality must occur within a time frame (in contrast with 'stative'). A further distinction is made between atomic events (achievements) and complex events (accomplishments). The first are instantaneous and the last are actions with a time duration.

Because of the subdivision of an event in DOLCE into atomic and complex events, an event in DOLCE represents either an ***achievement*** or an ***activity***.

5.1.2 Unified Foundation Ontology

UFO (Unified Foundation Ontology) is, like DOLCE, a foundational ontology. It describes entities in a domain independent way. Events in UFO are also defined as being a perdurant. More specifically an event "is a perdurant that is related to exactly two states (its pre-state and its post-state)". Like DOLCE a further distinction is made between an atomic event and a complex event. An atomic event is an instantaneous happening, while a complex event is composed of two or more events, which implies that the whole complex event requires a time duration.

[1] A perdurant is an entity that consists out of several temporal parts. A perdurant is only partially present in one moment in time. For example, the past and future temporal parts of a perdurant are not present. The converse of a perdurant is an endurant, which is an entity that is wholly present at any moment in time.

Events in UFO are either ***achievements*** or ***activities***.

5.1.3 Bunge-Wand-Weber

The BWW model is an ontology presented by Bunge [4] and extended by Wand and Weber [37, 36] to apply the ontology to information systems. They present a number of concepts that are supposed to represent all things in the real world, of interest to the information system modeller. An event in BWW is defined as a change of the (BWW-) state of a (BWW-) thing. Because an event in BWW represents a change, it can be classified in our notion of an ***achievement***.

5.1.4 The enterprise ontology

The Enterprise Ontology [33] is primarily developed to define the terms used to model an enterprise-wide view of an organisation. Every term needs to be defined explicitly, to create a shared understanding for every involved party. The primary function of the enterprise ontology is thus communication. As a second and third use, the authors define a stable basis for understanding and specifying the requirements and to achieve interoperability for different tools within an enterprise.

The ontology sets it's focus on the central term 'activity'. An event is defined as being a special kind of activity. This is however the only definition given. The authors leave the further interpretation of an event to the user. We conclude that an event can be classified as an ***activity*** in our model.

5.1.5 REA

The Resource-Event-Agent (REA) ontology [3, 11, 13] defines two kinds of events, economic events and business events. The economic events reflect changes in scarce means, while a business event represents a business activity that management wants to plan, monitor and evaluate, but doesn't result in a change in resources. An event in REA represent an ***activity***.

REA can be used to describe the bill-of-material of an enterprise. It can be used to build a data-model for the enterprise. In this point of view, the events defined in the model will be translated to activities as data concepts. They will become objects in a database, describing the properties and relations of these events.

Besides the description of the bill-of-material, business events in REA can also be used for activity based modelling.

5.2 State based modelling

5.2.1 UML

UML is not a modelling technique, but the concept of event and it's use is defined in the UML language specifications. Events in UML are used in the state machine package [17]. They are defined as:

> "[...] a result of some action either within the system or in the environment surrounding the system. An event is then conveyed to one or more targets. [...]"

Events are used to model interactions and behaviour of individual entities. When an event 'happens', it can trigger transitions in a state machine.

An event in UML can represent *a*) the occurence of an action; *b*) the occurence of a change; *c*) the expiration of a deadline; and *d*) the reception of a request (not the request itself).

Because of the interaction nature of events in UML and because an event in UML represents a happening at one point in time, events in UML are used as ***achievements***.

5.2.2 Syntropy

Syntropy [7] is a modelling technique constructed around the concepts 'object', 'value' and 'event'.

Events are used to model the state changes in the information system. A distinction is made between real world events and the (message) events that happen in the information system. An event in the real world is a happening (eg. a new employee joins the company). This real world event has consequences in the information system (a new employee needs to be registered). To achieve this consistency between the real world and the information system, a message event is created in the information system, saying that something needs to be done (add a new employee). This information system event can be received by all objects in the system, which will act accordingly. An overview of this relation is shown in figure 5.

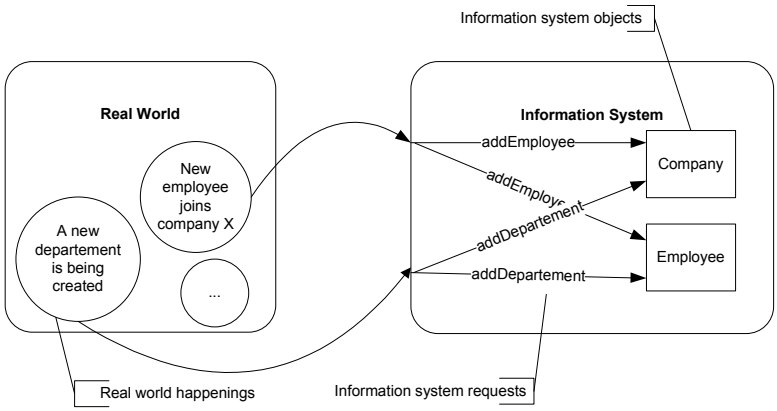


Figure 5: Coupling between real world events and information system events in Syntropy

Information system events in Syntropy thus serve to keep the consistency between the real world and the information system. The event in the information system has a consequence. It can bring objects in the model into existence, cause objects in the model to leave it and change properties of existing objects. Events *trigger* the state transitions of objects. These are separate operations defined in the objects.

Events in Syntropy express a need in the information system, which can be caught by the objects in the system. The event can also be seen as a notification of a real world event, but when viewed from an information system perspective, events in Syntropy are ***requests***.

5.2.3 Catalysis

Catalysis [10] is a modelling technique that uses sequence charts to model the interaction between entities and state charts to model the individual behaviour of entities. The outcome of actions in Catalysis are requests to execute other actions. A simple request in Catalysis is modelled as a Request/Reply. The concept event is only used when this request must be done asynchronously. Events are not used as a basic building block, they can only help the developer when needed.

Events are used as ***requests***. They take care of the (asynchronous) interaction between actions.

5.2.4 Merode

Merode [32] uses events as a basic building block. The technique defines three basic models: an existence dependency graph containing objects, a finite state machine, modelling the behaviour of objects and an object-event-table, modelling the interaction between events and objects. In the object-event-table each object type defines which method should be executed when an event *happens*, they describe the effect of an arriving event. In the finite state machine, the transition

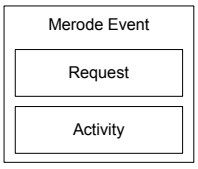


Figure 6: Merode Event

between states is also modelled with the help of methods (not events).
Events in Merode take care of the interaction between objects (they can be dispatched and caught) and the methods in the objects take care of the state transitions (the effect of an event). In this context Merode events are ***requests***.

A special note has to be made. It is possible to view events in Merode at a higher abstraction level. This way, the concepts of event and method in Merode are merged, which also implies that an event is a request (the interaction between objects) *and* an activity (the state transitions).

5.2.5 Object Oriented SSADM

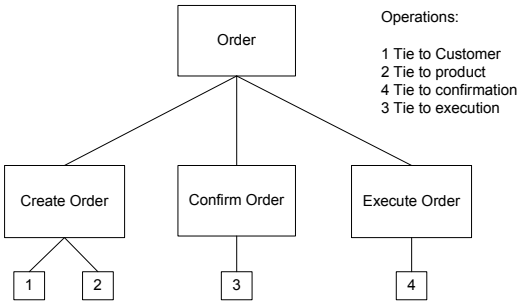


Figure 7: Object Oriented SSADM

Object Oriented Structured Systems Analysis and Design Method (SSADM) [29] also uses events to model the behaviour of individual objects. Like in Merode and Syntropy, an event in SSADM represents a need in the information system. An event is a request to execute certain actions, or a request to move the system to a next state. In SSADM the modeller defines a sequence of possible events for each object and for each event, operations are defined that are triggered by the event.

Events in SSADM are ***requests***.

5.3 Activity based modelling

5.3.1 Agent-Object-Relationship

The AOR model [35] is a modelling technique that sets its focus on *agents* and *objects*. Agents are introduced in the modelling grammar to capture the dynamic aspects of an information system. Only agents can perform actions, perceive events, communicate with one another and interact with objects. In the AOR model, events are used to indicate actions, either an action from an agent within the model, or an action not produced by an agent (e.g. from the environment). The first one is called an action-event and the latter is called a non-action event. Agents can perform action-events and other agents can perceive these events.

Events in AOR are ***activities***.

5.3.2 Event driven process chains

Event driven process chains (EPC) [9, 30, 34] are used for process modelling. They describe the process as a flow of activities. The connection between activities is realised by using events. Each activity must be preceded by an event and must be succeeded by an event. The preceding event is a precondition for the activity and the succeeding event is the postcondition for the activity. Events thus indicate changes in the information system (e.g. order created). They represent the end of an activity and the change that corresponds with this.

EPC also defines composition of events. Events can be refined in finer grained events. An event 'order checked', can for example be fragmented in the events 'customer data checked' and 'product data checked'.

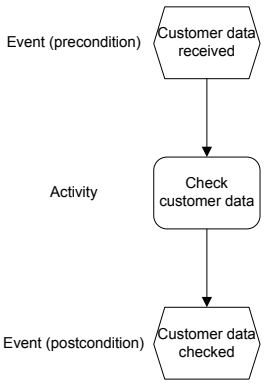


Figure 8: Event driven process chain

Because events are used as preconditions and postconditions, they signify something that already happened, something that is achieved or changed. Events in EPC are ***achievements***.

5.3.3 Process-driven event based business object model

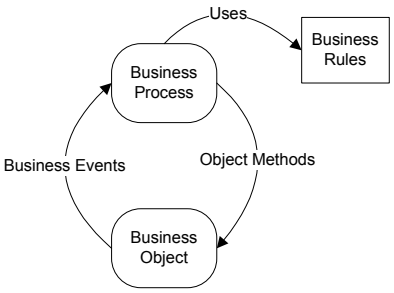


Figure 9: Process-driven event based business object model

In [28] a method is described to enhance the coupling between the IT system and the real-world business. The approach argues that most IT systems are very static and most of the time, the real world business must adapt to the (existing) information system. This is not the desired situation. The IT system must be able to adapt quickly to changing business demands. This is realised by introducing events in the methodology. Events are explicitly used to enhance loose coupling and flexibility.

A business event in the methodology indicates a state change in a business object. This event can trigger a business process, which uses business rules to help with the execution of the business process. The execution of a business process can in turn result in state changes of business objects (see figure 9).

Events in this methodology indicate state changes, they are ***achievements***.

Notice the difference in use between this technique and the events used in state machines (section 5.2). In state machines, events trigger state changes in objects, while in this technique, events are the state changes, they trigger business processes instead.

5.3.4 Remora

Remora [6] is a process modelling technique that uses events to trigger operations. An event in Remora indicates a state change in an object. These events trigger operations and an operation changes the state of the objects. The difference with the above technique is that not all state changes result in events. An event is only send out when the state change triggers a determined operation. For example: a stock change is only an event when this stock change must result in a restock order.

An event thus *always* triggers an operation in Remora, otherwise it isn't an event. In this retrospect, events in Remora are ***Requests***.

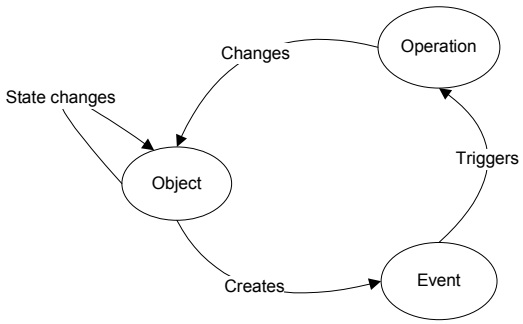


Figure 10: Remora

5.3.5 EDOC

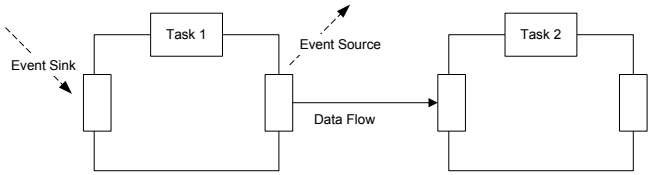


Figure 11: EDOC

Enterprise Distributed Object Computing (EDOC) [18, 2] is a modelling technique specially designed for distributed computing. A business process in EDOC is represented as a graph of business tasks linked in a specific order. Each business task defines an input and output. A task can receive input from a data flow, control flow or it can be triggered by an event. The event model in EDOC is used to accomplish asynchronous broadcast to other elements in the model. ***Achievement*** events are used in EDOC, they signify the actions or state changes that need to be exposed to other elements in the enterprise. A nuance has to be made with this proposition. EDOC doesn't clearly define which event concept should be used. According to the specifications, events should be notifications of actions or state changes, but nothing prevents the modeller to use *request-events*.

EDOC also defines a link with the implementation layer, more specifically with CORBA events [1].

5.3.6 Dynamic Workflow Modelling

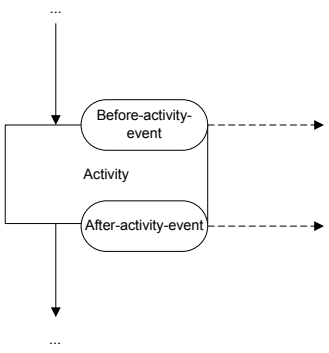


Figure 12: Dynamic Worklfow Modelling

Dynamic workflow modelling [22] defines a workflow model where an event is used as a basic building block to enforce decoupling between requester and receiver. Like EPC, they define for each activity a before-activity-event and an after-activity event. The first signifies that the activity starts and the latter signifies that the activity has ended. Each event can trigger business rules (with the

format: condition - action - alternative action). The difference with EPC is that the uses of the two events per activity isn't enforced.

Events are used as ***achievements***.

5.3.7 BPMN

BPMN (version 1.1 [39, 15] and RFP version 2.0 [16]) describes an event as something that *happens*. The event can be the trigger to start or alter the process flow, or it can be the action that happens when the process ends. BPMN doens't describe *how* the event triggers the process, the notation just states that the process starts/ends/alters when something happens. This *something* can be the catching or throwing of a message, signal, exception, ... For example: a timer start event indicates that the process must start at 9p.m. The notation says that the process starts when this happens.

Therefor, an event in BPMN is a happening in time (catch or throw of a signal), events in BPMN are ***achievements***. An event in BPMN is also very low level (it is a catch or a throw of something), it doesn't compare to the real-world business events.

5.3.8 BPDM

The business process definition metamodel (BPDM) [14] provides a common basis for process oriented models. It is a metamodel that can be used by others (e.g. BPMN). Events in BPDM are defined as happenings. Events denote a *change*. An event in BPDM can only occur at one moment in time. A separate concept is defined for 'a happening over time'. Because events in BPDM represent happenings at one point in time, we can classify them as ***achievements***

5.4 Implementation of the information system

In the implementation layer, events also have different meanings. The standard use of the concept event in the implementation layer, is an event as an observable happening in the system [25]. This equals our definition of an achievement. Besides this definition of an event, notifications are also defined. A notification is the message that flows from Alice to Bob indicating that something happened, it is the message containing all the necessary information of the event (achievement) that is observed. The two concepts live side by side. An event (happening) can be observed in the system and notifications that describe this event can be produced and broadcast (and eventually consumed). These definitions are used in publish/subscribe architectures (e.g. in Rebeca [24], Siena [5], JEDI [8], ...) and in active database systems [27].

However, the separation of notifications and (achievement) events is not used in the literature describing complex event processing (CEP) [21, 20]. In CEP an event is defined as follows: *An event is an object that is a record of an activity in a system. The event signifies the activity.* The event is not the activity itself, it describes the activity. In CEP the concept of event is used for a *notification* (there is no separate concept for a notification).

Another use of the concept event in the implementation layer is in BPEL [26] where events are used as *requests*. BPEL defines an event as a request-message from a *defined* partner that can be received during the course of the process. These 'events' are not events in the sense of a throw and catch mechanism, or a one to many communication. An event in BPEL is always an interaction between two pre-defined participants (no broadcast). The only difference with a request/reply, is that the event can occur anywhere in the process (not at a predefined point).

5.5 Other Differences

Below we describe the other differences of the use of the event concept. This is only done for the requirement engineering techniques. Ontologies only give definitions and relations for certain entities, they don't define their use (broadcast, loose coupling, composition, ...), with the exception of REA. REA defines the use of business events for activity based modelling, thus is included in the analysis.

5.5.1 Abstraction level of events

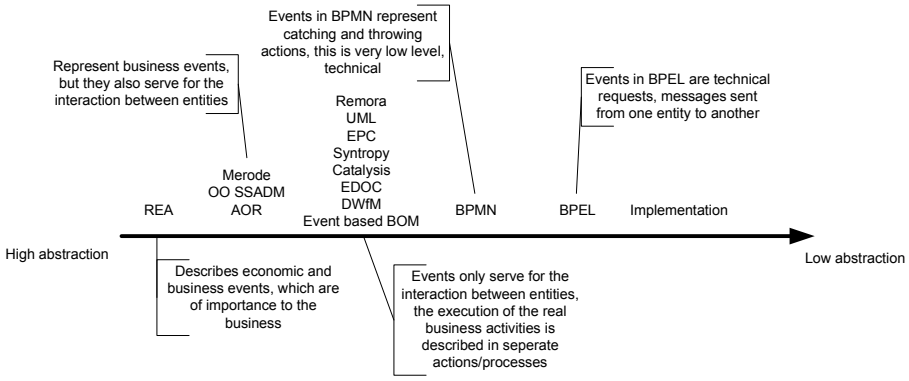


Figure 13: Abstraction levels of events in the different RE techniques

In figure 13 an ordering is given for the use of the concept event for the different techniques. The scale is described in section 4.2. The requirements engineering techniques are ordered by the level of abstraction that the modeller has to think about, when defining events in the technique. High on the scale means more business related events, low on the scales means more detailed, technical events.

5.5.2 Composition, use and dispatching of events

	State charts				Process Modelling
	UML	Syntropy	Catalysis	Merode	EPC
Composition of events				X	X
Intended for loose coupling explicitly mentioned			X		
Dispatching		Broadcast	Defined or broadcast	Defined	

	Process Modelling				
	EDOC	Remora	DWfM	BPMN	event based BOM
Composition of events					
Intended for loose coupling explicitly mentioned	X		X		X
Dispatching	Broadcast			Defined or broadcast	Broadcast

	Process Modelling			
	OO SSADM	AOR	Implementation	REA
Composition of events			X	
Intended for loose coupling explicitly mentioned			X	
Dispatching	Defined		N/A	

Table 2: Composition, use and dispatching of events

The use of events in the modelling techniques also differ in the possibility of composition, intended use and dispatching of events (see section 4.2). In table 2 an overview is given for each technique. For the *composition of events*, an X means that it is mentioned in the technique that events can be composed out of other events. For the *intended use*, an X means that it is explicitly mentioned in the technique that the concept event is used to enhance loose coupling or distributed computing. *Dispatching* determines the way events are dispatched. Events can be broadcast to every entity in the model, or the receivers of an event can be predefined. When the cell is empty, it is not mentioned how the dispatching happens.

5.6 Conclusion of event interpretations

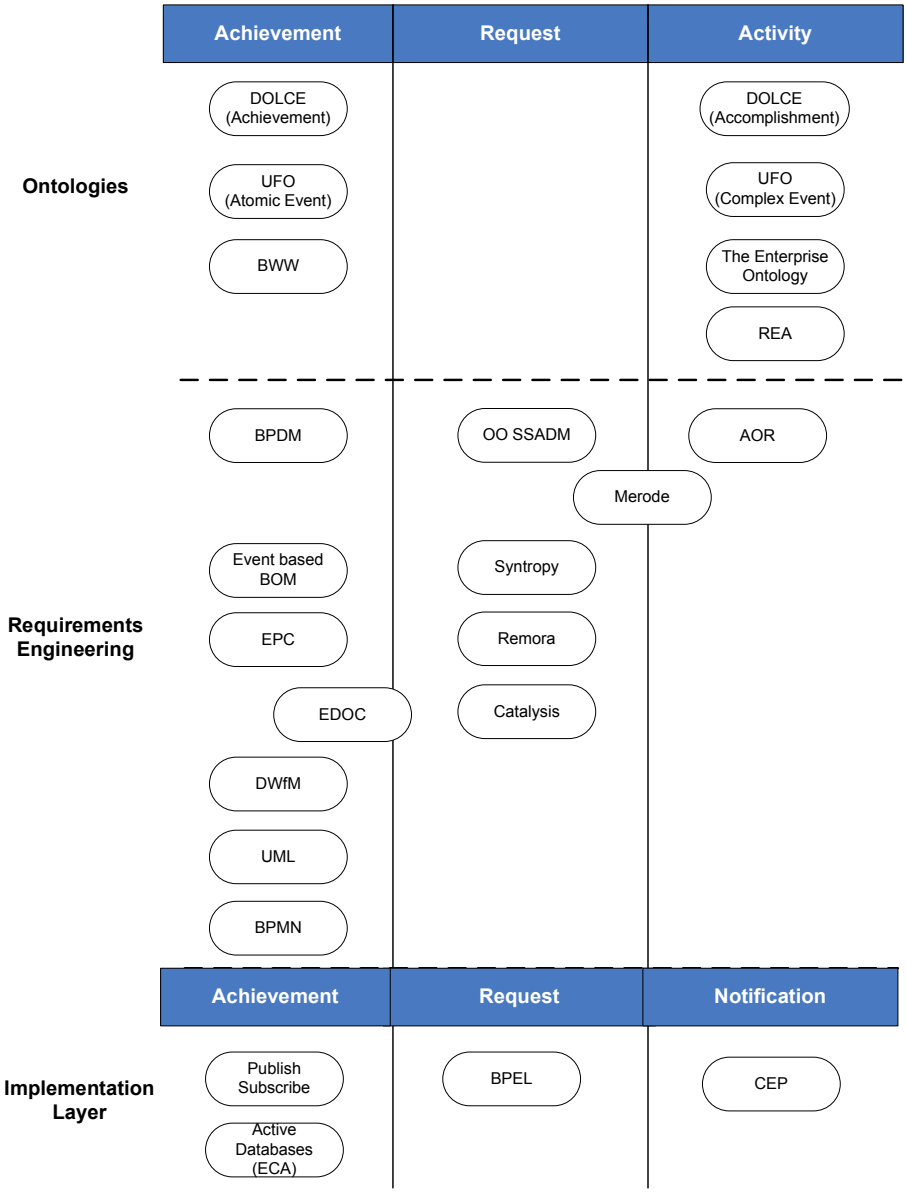


Figure 14: Event Interpretations conclusion

In figure 14 an overview is given for the different requirements engineering techniques and their interpretations of the concept event. As a vertical scale, the abstraction scale of events as defined in section 5.5.1 is used. This vertical abstraction scale is not absolute, for a correct comparison, look at figure 13. The emphasis in figure 14 is on the classification of the techniques.

Most definitions use the concept of event for interaction between elements. In requirements engineering an event is something that happens. Other activities/agents can observe this happening and react to it. Most techniques use the achievement event for this interaction. This corresponds also with the use of events in the implementation of the information system. Few techniques use the notion of a request. However, most of the time the difference between an achievement and a request is very unclear and not well defined. Only thorough investigation of the description of the techniques and examples make the use of 'an event' more clear (achievement or request).

A fairly good consensus is reached in the enterprise ontologies about the concept event. An event is defined as either an achievement or an activity. UFO and DOLCE make a distinction between atomic events and complex events. The use of event as activity is however not widely used in the requirement

engineering techniques. An explanation for this can be that RE models define the interaction between activities or processes. An event defines when a process should start or end. This can't be done when the event is a happening with a time duration. If so, when should the process start? A process or activity always starts at one point in time. When achievement-events are used, this point in time can be precisely defined.

6 Conclusion

The concept event is used for many purposes, and it is not always clear which purpose is intended. When analysing the different requirements engineering techniques, it was difficult to find out what each technique meant with an event. Except for the ontologies, each technique gives a rather simplistic definition of an event. Especially the difference between an achievement and a request is not clear. Even within one technique achievements and requests are used in parallel (e.g. in EDOC). This all can lead to confusion, because the achievement and request concepts still require a different line of thought ('what happens / what happened' vs 'what must happen').

6.1 What is missing?

Well defined concepts

Link with economic / business events The link between the real world economic or business events is missing in the requirements models. Events are used to model interaction between entities. They are not the (economic) happenings itself. This way it is hard to track which economic event triggered which information system event.

Link with implementation A link with the implementation layer is also missing. No rules are defined on how to convert events defined in the model to an event driven architecture in implementation. Most of the time the same concept for an event is used in the requirements models and in implementation (an achievement), but not always. Transformation models should be defined on how to convert events to executable actions or notification, ...

Causality Causality between events is a strong tool for analysing events [21]. The causality between events should already be defined in RE, so that it can be used in the implementation layer. Causality between events, however, is mostly missing in RE literature. State-machines for example represent a sequence of events, but don't necessarily define a causal relationship.

Vertical causality over the different information system development layers can for example solve the requirements traceability problem. Requirements traceability means that requirements should be tracked throughout the different software development layers. When something happens in the implementation layer, we can indicate that it happened because of a specific requirement. With the help of vertical causality between technical events and RE events, an event at the lowest level (implementation), can be traced back to the enterprise modelling layer. For example, the technical events *placed-on-fabrication-line*, *assembled*, *painted*, *dried*, *left-fabrication-line* happened, because there is a requirements event *car-created*. Business users also rather know that this last RE event happened, they don't care for the underlying, 'difficult to understand' technical events.

6.2 Future Research

To resolve the 'what's missing' problems, we can start with investigating recommendations for the use of the three event interpretations. Should these event concepts live side by side? Should only one concept be used? When should we use which concept? Afterwards a link (and transformation rules) with the implementation of the information system can be defined. At this stage causality between different events can be outlined. Horizontal causality (e.g. the execution of an activity is causally

dependent on a request) as well as vertical causality which accomplishes a link between technical events and higher level requirement-events. This last stage brings us back to the link with real world economic events.

References

[1] S. Abraham, K. Duddy, M. Lawley, Z. Milosevic, K. Raymond, and A. Wood. Mapping enterprise events to the corba notification service. *Enterprise Distributed Object Computing Conference, 2000. EDOC 2000. Proceedings. Fourth International*, pages 124–134, 2000.

[2] A. Barros, K. Duddy, M. Lawley, Z. Milosevic, and A Wood. Processes, roles, and events: UML concepts for enterprise architecture. *Lecture notes in computer science*, pages 62–77, 2000.

[3] M. Bergholtz, P. Jayaweera, P. Johannesson, and P. Wohed. Process models and business models-a unified framework. *Lecture notes in computer science*, pages 364–377, 2003.

[4] M.A. Bunge. *Treatise on Baisc Philosophy: Volume 3: Ontology 1: The furniture of the world.* Reidel, 1977.

[5] A. Carzaniga, D. S. Rosenblum, and A. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, 2001.

[6] R. Colette. A methodology for information system design. In *AFIPS National Computer Conference*, pages 583–589, 1981.

[7] S. Cook and J. Daniels. *Designing object systems: object-oriented modelling with Syntropy.* Prentice Hall, 1995.

[8] G. Cugola, E. Di Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *Software Engineering, IEEE Transactions on*, 27(9):827–850, Sep 2001.

[9] R. Davis and E. Brabnder. *ARIS Design Platform.* Springer London, 2007.

[10] D.S. Desmond Francis and A.C. Wills. *Objects, components, and frameworks with UML: the catalysis approach.* Addison-Wesley, 1999.

[11] F. Gailly and G. Poels. Ontology-Driven Business Modelling: Improving the Conceptual Representation of the REA Ontology. *Lecture Notes in Computer Science*, 4801:407, 2007.

[12] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening ontologies with DOLCE. *Lecture notes in computer science*, pages 166–181, 2002.

[13] G. Geerts and W.E. McCarthy. An accounting object infrastructure for knowledge-based enterprise models. *IEEE Intelligent Systems and Their Applications*, 14(4):89–94, 1999.

[14] Object Management Group. Bpdm 1.0, omg specifications. `http://www.omg.org/spec/BPDM/1.0/`, June 2009.

[15] Object Management Group. Bpmn 1.1, omg specifications. `http://www.bpmn.org/Documents/BPMN1-1Specification.pdf`, May 2009.

[16] Object Management Group. Draft proposal for: Bpmn 2.0. `http://www.omg.org/cgi-bin/doc?bmi/2007-6-5`, May 2009.

[17] Object Management Group. Uml omg specifications. `http://www.omg.org/technology/documents/modeling_spec_catalog.htm\#UML`, May 2009.

[18] Object Management Group. Uml profile for enterprise distributed object computing. `http://www.omg.org/technology/documents/formal/edoc.htm`, May 2009.

[19] A.G. Kleppe, J. Warmer, and W. Bast. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003.

[20] D.C. Luckham and J. Vera. An event-based architecture definition language. *Software Engineering, IEEE Transactions on*, 21(9):717–734, Sep 1995.

[21] D. Luckman. *The power of events: an introduction to Complex Event Processing*. Addison-Wesley, 2002.

[22] Jie Meng, S.Y.W. Su, H. Lam, and A. Helal. Achieving dynamic inter-organizational workflow management by integrating business processes, events and rules. *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, page 10, Jan. 2002.

[23] P.A.P. Moran. *An introduction to probability theory*. Oxford University Press, USA, 1984.

[24] G. Mühl. Large-scale content-based publish/subscribe systems. *PhD thesis, Darmstad University of Technology*, 2002.

[25] G. Mühl, L. Fiege, and P. Pietzuch. *Distributed Event-Based Systems*. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2006.

[26] Oasis. Web service business process execution language version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`, May 2009.

[27] N. W. Paton and O. Díaz. Active database systems. *ACM Comput. Surv.*, 31(1):63–103, 1999.

[28] K. Riemer. A process-driven, event-based business object model. *Enterprise Distributed Object Computing Workshop, 1998. EDOC '98. Proceedings. Second International*, pages 68–74, Nov 1998.

[29] K. Robinson and G. Berrisford. *Object-oriented SSADM*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1994.

[30] A.W. Scheer, O. Thomas, and O. Adam. *Process Modeling Using Event-Driven Process Chains*. Wiley-Interscience, 2005.

[31] J.G. Siegel and J.K. Shim. *Dictionary of accounting terms*. Barron's Educational Series, Inc., 2005.

[32] M. Snoeck. *Object-oriented enterprise modelling with MERODE*. Leuven University Press, 1999.

[33] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. *The knowledge engineering review*, 13(01):31–89, 1998.

[34] WMP Van der Aalst. Formalization and verification of event-driven process chains. *Information and Software technology*, 41(10):639–650, 1999.

[35] G. Wagner. The Agent–Object-Relationship metamodel: towards a unified view of state and behavior. *Information Systems*, 28(5):475–504, 2003.

[36] Y. Wand and R. Weber. An ontological model of an information system. *IEEE transactions on software engineering*, 16(11):1282–1292, 1990.

[37] Y. Wand and R. Weber. On the deep structure of information systems. *Information Systems Journal*, 5(3):203–223, 1995.

[38] R. Weber. *Ontological foundations of information systems*. Coopers & Lybrand Accounting Research Methodology, 1997.

[39] S.A. White. Introduction to BPMN. *IBM Cooperation*, pages 2008–029, 2004.